Amendments to the Specification

Please replace paragraphs [0015], [0017], [0052], [0053], [0060], and [0061] with the following amended paragraph:

[0015] A third approach to distributed computing is distributed user interface toolkits, which address the issues that arise when employing web-based HTTP/HTML and remote frame buffer approaches by allowing a server to manipulate user interface toolkit components directly on the client. The server can create, modify, and delete any of the components available in the distributed toolkit as if it were working with a local application. This approach is analogous to an implementation of a remote frame buffer with an extremely efficient, lossless compression algorithm. Instead of sending pixel data rendered on the server across the network, the distributed user interface toolkit sends the semantics necessary to render that pixel data on the client. In addition, since the mouse is handled locally on the client, there is no additional perceived latency beyond that caused by the processing that is necessary to service ~~users~~ users' requests when the application is running locally.

[0017] There is therefore a need in the art for a distributed user interface that runs the application logic on the server computer but which also ~~allow~~ allows the server computer to asynchronously generate events and transmit them to the server. There is also a need for a distributed user interface that allows relatively sophisticated graphics without requiring high-bandwidth connections. In addition, there is also a need for a distributed user interface which is easily implemented and does not require the creation of a new protocol of communication.

[0052] A next stage in the procedure may be to invoke the remote-capable user interface toolkit 108 by the application logic 106 according to the application programming interface to perform a function. At a subsequent stage, the remote-capable user interface toolkit 108 generates a remote message to perform the function invoked by the application logic 106. Since there is a one-to-one correspondence between the JFC component and the RJFC component, a protocol of communication between the RJFC component and the JFC component is implicitly defined. This protocol of communication comprises the transferring of ~~massages~~ messages to perform JFC functions, and such messages are issued in the manner in which the JFC toolkit would normally perform functions on a single computer. Therefore, there is no need to specifically create a protocol of communication.

[0053] The message may be communicated between the remote-capable user interface toolkit on the server and the user interface toolkit on the remote client in a subsequent step. In the exemplary embodiment, this communication between the server 102 and the client 104 uses remote method invocation (RMI) (See S. McPherson, "JAVA™ Servlets and Serialization with RMI," http://developer.java.sun.com/developer/technicalArticles/RMI/rmi/.)

[0060] FIG. 8 illustrates a simple spreadsheet application 240, ~~each of~~ which is rendered on the client buffer 116 in response to commands generated by the RJFC toolkit 118 in accordance with the invention.

[0061] An example of code for creating a simple "notepad" application written using the RJFC API is shown in FIG. 9(a), and a baseline, i.e., non-network-aware, version of the code in JFC API is shown in FIG. 9(b). The code generator was configured such that the resulting RJFC API has a one-to-one correspondence to JFC components. In the exemplary embodiment, a capital "R" (indicative of the remote-capable functionality) is prepended to the name of the toolkit component being referenced. The significant difference between the non-network-aware JFC application of FIG. 9(b) and the remote-capable RJFC application of FIG. 9(a) is that the JFC code calls "new" to instantiate a component, whereas the RJFC code makes a remote method invocation to an RJFCFactory object (as described above) which resides in the client viewer's memory space.